UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

ALPR: Sistema de reconocimiento automático de números y caracteres en matrículas

Autor/es

DARÍO PASCUAL MORALES

Director/es

ÁNGEL LUIS RUBIO GARCÍA

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2016-17

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

ALPR

Sistema de reconocimiento automático de números y caracteres en matrículas

Autor: Darío Pascual Morales
Tutor: Ángel Luis Rubio García
Curso: 2016-2017

Grado en Ingeniería Informática
Departamento de Matemáticas y Computación
Facultad de Ciencia y Tecnología
Universidad de La Rioja

Resumen

A finales de los años 60, en las universidades pioneras en investigaciones de inteligencia artificial, surgió una rama de esta conocida como visión artificial o visión por computador.

Desde ese momento, esta disciplina se centró en desarrollar métodos para adquirir, procesar, analizar y comprender imágenes con el fin de producir información numérica o simbólica que pueda ser tratada por un ordenador.

Uniendo las técnicas de procesamiento de imágenes a las técnicas de aprendizaje automatizado, podemos desarrollar programas capaces de clasificar imágenes o extraer unas determinadas características de las mismas. Para que estas técnicas de aprendizaje supervisado funcionen, es necesario entrenar al ordenador proporcionando patrones previamente etiquetados, de forma que el algoritmo pueda así crear las fronteras y separar a los grupos de la forma deseada.

La aplicación cuyo desarrollo está explicado a lo largo de este documento junta tanto la visión por computador como las técnicas de aprendizaje automatizado. Gracias a esto, es capaz de extraer, tanto de fotografías como de vídeos, la información contenida en las matrículas de los vehículos que aparecen en ambas fuentes de imagen.

Los algoritmos de visión artificial se encargan de localizar la matrícula y los caracteres de la misma en toda la imagen. Por otro lado, gracias al aprendizaje automatizado y más concretamente a las Máquinas Vector Soporte (SVM), podemos clasificar los caracteres anteriormente extraídos y almacenar la matrícula reconocida en formato textual.

Para concluir esta introducción, me gustaría dejar claro que soy consciente de que el reconocimiento de matrículas no es algo novedoso, pero teniendo en cuenta la enorme cantidad de imágenes que cada día se suben a internet (añadiéndose a la infinidad de imágenes que ya hay) y la importancia que está cobrando en la informática actual el aprendizaje automatizado, este proyecto me ha parecido una buena forma de entrar en contacto con el procesamiento y la clasificación automática de imágenes, ya que estas técnicas no se aprenden durante la etapa universitaria.

Summary

In the late 1960s, in the pioneering universities in artificial intelligence investigation, a branch known as artificial vision or computer vision came up.

Since that moment, that discipline focussed on developing methods for acquiring, processing, analysing and understanding images to produce numeric or symbolic information which could be treated by a computer.

Joining the techniques of image processing with machine learning techniques, we can develop programs capable of classifying images or extracting their characteristics. For these supervised learning techniques to work, it is necessary to prepare the computer providing previously labelled patterns so that the algorithm could create the borders and separate the groups as desired.

The application, whose development is explained in this document, combines both computer vision and machine learning techniques. Thanks to this, it is capable of extracting, both from photographs and videos, the information contained in the licence plates of the vehicles that appear in both image sources.

The artificial vision algorithms oversee the location of the licence plate and its characters in all the images. On the other hand, thanks to the automatized learning, and more specifically to Support Vector Machines (SVM), we can classify the previous characters and store the licence plate recognised in textual format.

In conclusion, I would want to make it clear that I am aware of that the licence plates recognition is not new, but taking into account the amount of images that are shared on the internet every day (combining with the images that are already shared) and the importance of the automatized learning in computing, I thought this project would be a good way to get in contact with the processing and automatized classification of images, since these techniques are not studied in the university stage.

Índice

Índice de figuras

Capítulo 1. Introducción.

1 INTRODUCCIÓN

1.1 MOTIVACIÓN

Es habitual que los accesos a los parkings y autopistas estén controlados por cámaras de video. Los sistemas de extracción de información de videos e imágenes cada vez son más comunes, por lo que interesa mantenerlos al día haciendo uso de mejores algoritmos. Además, existe una motivación personal ya que nunca antes he trabajado con librerías de procesamiento de imágenes.

Este proyecto surge con la motivación de facilitar la gestión de las entradas y salidas de vehículos en parkings y autopistas. Para ello, se ha pensado en el desarrollo de un programa que detecte las matrículas de los vehículos grabados por las cámaras para, posteriormente, reconocer los caracteres contenidos en las mismas.

He de destacar que este Trabajo de Fin de Grado se desarrollará en colaboración con la empresa Formavolucion S.L.

Un tercer ente está implicado en este proyecto. Una empresa que tiene la intención de comercializar el programa en Brasil, por lo que todos los requisitos del sistema de detección girarán en torno a las matrículas de dicho país. Las imágenes necesarias para testear la aplicación y realizar los entrenamientos serán proporcionadas por esta empresa.

1.2 OBJETIVOS

1.2.1 *Objetivos del proyecto*

El objetivo del proyecto es desarrollar una aplicación de escritorio con la que se consiga extraer de los videos proporcionados por las cámaras una captura de las matrículas, los caracteres reconocidos en la misma en formato textual y la hora de entrada.

Con esto se evita la necesidad de almacenar los videos completos y se facilita la obtención de estadísticas de acceso o la consulta de un acceso concreto.

Otro objetivo deseable es que la aplicación sea eficiente en tiempo, ya que la barrera que proteja el acceso al parking o a la autopista no se levantará hasta que el reconocimiento no sea completo.

Esta aplicación será utilizada para realizar las demostraciones del funcionamiento del sistema de reconocimiento a futuros clientes. Esto justifica el desarrollo de una aplicación

de escritorio, dado que desarrollar esto en un entorno web añadiría al proyecto una complejidad innecesaria.

Para que exista la posibilidad de implantarlo como sistema final, la parte de la lógica de la aplicación será escrita, dentro de lo posible, en C++ estándar. Esto permitirá migrar la aplicación a sistemas Unix, los más comunes en los equipos encargados de gestionar estos accesos.

1.2.2 *Objetivos académicos*

Con la realización de este proyecto se espera:

- Adquirir experiencia de trabajo en proyectos reales y ampliar la experiencia en la gestión de dichos proyectos.
- Además de mejorar los conocimientos adquiridos durante toda la carrera de lenguajes como C# y C++, aprender a utilizar librerías de tratamiento de imágenes y aprendizaje automatizado.

Capítulo 2. Planificación

1 GESTIÓN DEL ALCANCE

1.1 PLANIFICACIÓN DEL ALCANCE

Como punto de partida del proyecto se ha intentado definir una serie de requisitos para el producto que se iba a desarrollar. En esta primera reunión se habló del posible uso de métodos de aprendizaje diferentes a los que se estaban utilizando hasta el momento o de la realización de un software multiplataforma.

Tras realizar un análisis de ambas propuestas, se ha considerado que, aunque sean buenas, deben quedarse para futuras fases del desarrollo y excluirse de los objetivos de este proyecto debido a las limitaciones temporales.

Introducir un nuevo método de aprendizaje supondría no poder apoyarme en el software con el que se cuenta en la empresa que, aunque desactualizado e ineficiente, puede ser un buen punto de partida para realizar mi programa de aprendizaje. Por otro lado, usando los métodos de aprendizaje automatizado contenidos en OpenCV (con los que he trabajado durante el periodo de prácticas) se podría dedicar menos tiempo al estudio de métodos de aprendizaje y realizar una aplicación más completa.

Para realizar un software multiplataforma real se deberían desarrollar tantos programas como plataformas y arquitecturas se quieran cubrir (Windows, Unix, arquitecturas de 32 bits, 64 bits…). Para facilitar en un futuro estas conversiones, la parte de la aplicación que contiene la lógica de aprendizaje y reconocimiento se desarrollará modularmente siguiendo los estándares de C++. Así, se podrán realizar compilaciones tanto en plataformas Windows como en sistemas Unix permitiendo compilar cada módulo como aplicación de consola o pudiendo añadirlos como lógica de una nueva aplicación de escritorio.

1.2 DEFINICIÓN DEL ALCANCE

El producto que se va a desarrollar será una aplicación de escritorio para ser mostrada a las empresas encargadas de la gestión de accesos a parkings o autopistas. Las tecnologías que se usarán para el desarrollo serán (la justificación se puede encontrar en el punto 4):

- OpenCV (en su versión de C++), una librería libre de visión artificial, que se usará tanto para los entrenamientos como para la localización de las matrículas y sus respectivos caracteres.

- Visual C#, herramientas proporcionadas por Microsoft para crear interfaces de usuario para Windows.
- MySQL, gestor de bases de datos donde se almacenarán los resultados del análisis de los videos (captura de la matrícula, su contenido, hora de entrada...).

Esta aplicación permitirá visualizar las distintas cámaras sobre las que se están aplicando los métodos de reconocimiento. También debe permitir la gestión de las mismas, es decir, en todo momento se podrán añadir nuevas cámaras o eliminar las existentes. Además, deberá permitir añadir videos pregrabados en lugar de cámaras para facilitar la demostración de su correcto funcionamiento.

Por otro lado, permitirá realizar consultas sobre la base de datos donde se almacenan los resultados del reconocimiento, permitiendo la visualización total de los datos o realizar filtrados por número de matrícula o fecha de acceso.

Por último, contará con un apartado de configuración, que permitirá realizar tantos entrenamientos como se quiera, modificar algún parámetro de los algoritmos encargados de los entrenamientos y algunas configuraciones relacionadas con los datos a almacenar (pasos intermedios de reconocimiento, nueva ubicación de los datos de entrenamiento).

Un esquema de la estructura de la aplicación sería el siguiente.

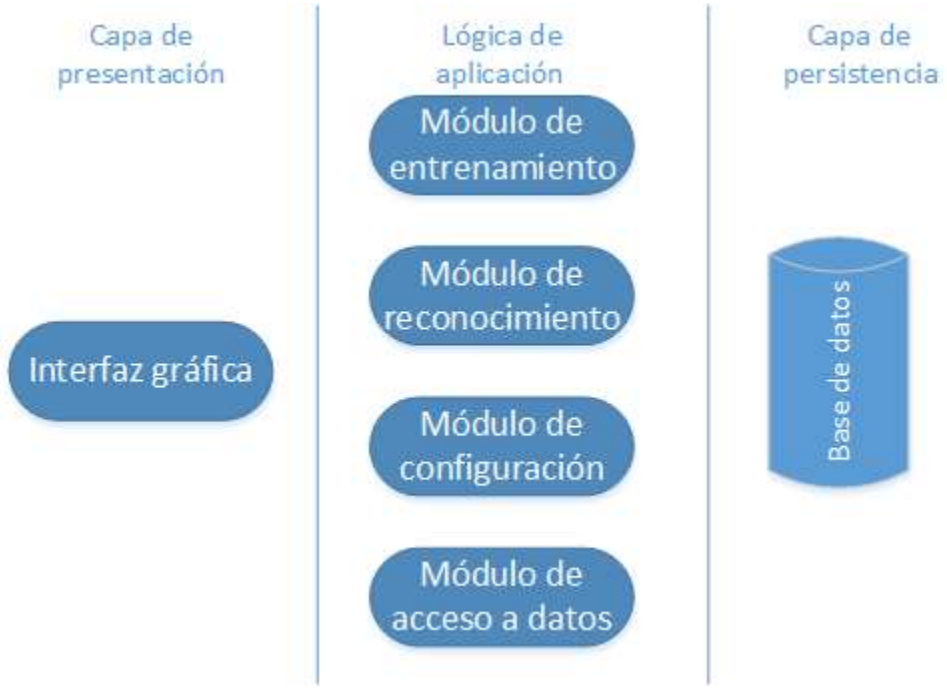

Figura 1. Estructura de la aplicación en tres capas

2 GESTIÓN DEL TIEMPO

2.1 PLANIFICACIÓN INICIAL DE TIEMPOS

Fase del proyecto	Número de horas
Definición del alcance y planificación temporal	5
Preparación del entorno de trabajo	15
Diseño de la interfaz de usuario	20
Desarrollo de la interfaz de usuario	20
Diseño de la lógica de negocio	40
Desarrollo de la lógica de negocio	80
Diseño de la capa de persistencia	15
Desarrollo de la capa de persistencia	20
Ensamblado y pruebas	30
Reuniones	5
Seguimiento del proyecto	10
Memoria y documentación	30
Diseño de la presentación del proyecto	10
Total	**300 horas**

Tabla 1. Estimación de tiempos

2.2 CALENDARIO DEL PROYECTO

En rojo los días festivos

Figura 2. Calendario del proyecto

Fase del proyecto	Fecha de inicio	Fecha de fin
Definición del alcance y planificación	01/02/2017	06/02/2017
Preparación del entorno de trabajo	07/02/2017	10/02/2017
Diseño de la interfaz de usuario	13/02/2017	17/02/2017
Desarrollo de la interfaz de usuario	20/02/2017	24/02/2017
Diseño de la lógica de negocio	27/02/2017	10/03/2017
Desarrollo de la lógica de negocio	13/03/2017	07/04/2017
Diseño de la capa de persistencia	10/04/2017	24/04/2017
Desarrollo de la capa de persistencia	25/04/2017	02/05/2017
Ensamblado y pruebas	03/05/2017	12/05/2017
Diseño de la presentación del proyecto	15/05/2017	17/05/2017
Depósito del proyecto	21/06/2017	23/06/2017
Presentación del proyecto	10/07/2017	12/07/2017

Figura 3. Leyenda del calendario del proyecto

Notar que las fases del proyecto correspondientes con el seguimiento del proyecto y la realización de la memoria no están incluidas en el calendario. Esto se debe a que serán dos tareas cuya realización será simultánea al desarrollo del proyecto.

Tampoco aparecen las reuniones ya que en esta fase del proyecto aún no se conocen las fechas concretas de las mismas. Otro aspecto a destacar es que los días previos a la preparación del proyecto aparece una tarea relacionada con el diseño de la presentación. Esta tarea se corresponderá con la preparación oral de la presentación.

Es muy probable que las fases de desarrollo de cada una de las capas no sean tan secuenciales como aparece en el calendario del proyecto y estén más interconectadas.

Las cuatro semanas entre la finalización teórica del proyecto y el depósito de este pueden ser de gran utilidad para realizar reajustes temporales de la planificación o evitar improvistos.

2.3 DETALLE DE LAS FASES DEL PROYECTO

- **Definición del alcance y planificación:** se especificará el alcance final del proyecto y se asignará un tiempo estimado a cada una de las fases del mismo. Como entregable de esta fase se obtendrá un documento de objetivos.
- **Preparación del entorno de trabajo:** instalación de MySQL, descarga e instalación de OpenCV e integración de ambas en Visual Studio.
- **Diseño de la interfaz de usuario:** realización de un prototipo de capa de presentación siguiendo aspectos formales del diseño de interfaces de usuario.

- **Desarrollo de la interfaz de usuario:** implementar la funcionalidad del prototipo con las herramientas proporcionadas por Visual C#.
- **Diseño de la lógica de negocio:** diseño formal de las clases de cada uno de los módulos que formarán la lógica de la aplicación.
- **Desarrollo de la lógica de negocio:** implementación de cada uno de los módulos integrando en ellos OpenCV. Además, se desarrollarán las clases que permitirán la compilación del módulo de reconocimiento y del de entrenamiento para ser ejecutados desde la consola.
- **Diseño de la capa de persistencia:** diseño formal de la base de datos.
- **Desarrollo de la capa de persistencia:** desarrollo de las clases encargadas de gestionar los datos dentro de la aplicación y creación de la base de datos en MySQL Workbench.
- **Ensamblado y pruebas:** testeo de cada uno de los módulos por separado y unificación de todas las capas de la aplicación.
- **Reuniones:** encuentros con el tutor para evaluar el seguimiento del proyecto.
- **Seguimiento del proyecto:** revisión de las horas planificadas respecto a las realmente utilizadas en cada una de las fases del proyecto. Se incluirá cualquier desviación temporal o replanificación.
- **Memoria y documentación:** elaboración del documento que contendrá los detalles del desarrollo del proyecto.
- **Diseño de la presentación del proyecto:** elaboración de una presentación que sirva de apoyo en la defensa del trabajo ante el tribunal.

3 RIESGOS

A continuación, se exponen una serie de riesgos que, presumiblemente, se van a afrontar durante el desarrollo del proyecto:

- **Aprendizaje de nuevas tecnologías:** como se ha comentado en el alcance, se va a utilizar OpenCV, una librería para el procesado de imágenes que, aunque usé durante el periodo de prácticas, fue para otros menesteres. Si el aprendizaje de la misma no es rápido, el tiempo de desarrollo puede crecer en exceso y puede que no se termine la aplicación por completo.
- **Planificación incorrecta:** a pesar de que, como se ve en el calendario del proyecto (Figura 2), hay tiempo para subsanar posibles imprevistos, si la planificación no es buena y la estimación de fechas es incorrecta puede darse el caso de que el proyecto no se entregue a tiempo ya que las fechas de entrega académica son estrictas.

4 DECISIONES Y JUSTIFICACIÓN

En este apartado se va a detallar porque se han hecho algunas elecciones previas, ya indicadas someramente en los apartados de alcance y planificación.

4.1 TECNOLOGÍAS

El uso de C++ para la lógica del proyecto es algo impuesto en las especificaciones previas del proyecto. Por lo que solo tengo libertad para decidir el diseño de la interfaz y la librería de procesamiento de imágenes.

- **Interfaz:** tras valorar lo visto a lo largo del grado, decido usar C#, ya que se integra a la perfección en Windows (sistema objetivo de la aplicación). Además, permite una alta velocidad de desarrollo.
- **Procesamiento de imágenes:** desde la empresa me recomiendan usar OpenCV, tanto para el procesamiento de las imágenes como para el aprendizaje automatizado.
  Al investigar sobre los métodos de aprendizaje de OpenCV descubrí que estaban apareciendo nuevas tecnologías para realizar estas operaciones. Entre todas, destacaban *Scikit Learn* y *TensorFlow*. Tras investigar ambas alternativas decido quedarme con OpenCV para todo, debido a que estas dos librerías están en estados iniciales de desarrollo (a fecha de febrero de 2017) y sus APIs para C++ son todavía muy precarias.
- **Persistencia:** a pesar de la aparente sencillez de la solución de persistencia decido usar MySQL para facilitar la consulta sobre los datos almacenados.

4.2 METODOLOGÍA

La decisión de planificar con un método tradicional en vez de que con metodologías más modernas (como Scrum, por ejemplo) se justifica porque es un proyecto que voy a desarrollar de manera individual y no se me va a requerir mostrar avances a ningún cliente en concreto.

Capítulo 3. Análisis de requisitos.

1 REQUISITOS FUNCIONALES

La funcionalidad de la aplicación se representa mediante el diagrama de casos de uso de la figura siguiente.

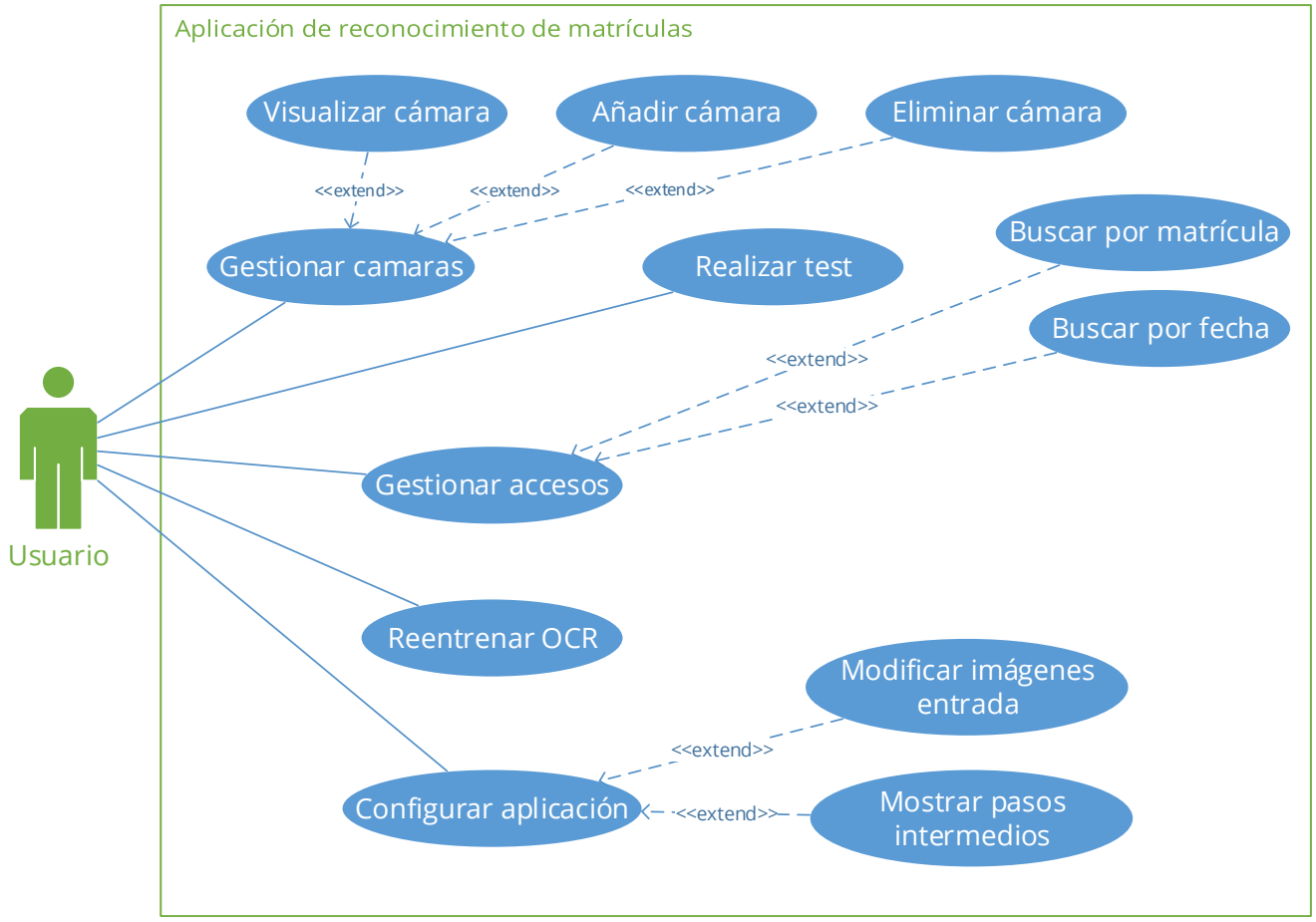

Figura 4. Diagrama de casos de uso

Una vez visto el diagrama, se definen los casos de uso que contienen la funcionalidad de la aplicación:

Caso de uso 1. Gestionar cámaras.

Actor: usuario.

1. El usuario pulsa sobre gestión de cámaras.
2. El sistema muestra todas las funcionalidades relacionadas con la gestión de las cámaras.

Caso de uso 1.1. Visualizar cámara.

Actor: usuario.

1. El usuario pulsa sobre el nombre de una de las cámaras añadidas.
2. El sistema mostrará la imagen en directo de dicha cámara.

Caso de uso 1.2. Añadir cámara.

Actor: usuario.

3. El usuario pulsa en añadir cámara y escribe la dirección de la misma y un nombre.
4. En la interfaz se añade un acceso a la nueva cámara.

Extensiones: si el usuario no introduce correctamente los datos, la aplicación mostrará un mensaje de error y se dará la opción de introducirlos de nuevo.

Caso de uso 1.3. Eliminar cámara.

Actor: usuario.

1. El usuario pulsa sobre eliminar cámara.
2. Se le solicita al usuario una verificación de la acción para evitar errores en la eliminación.
3. El acceso a la cámara eliminada desaparece de la interfaz.

Caso de uso 2. Realizar test.

Actor: usuario.

1. El usuario pulsa en realizar test y se le da la posibilidad de elegir un video de su disco duro.
2. Acepta el test y se muestra en pantalla el resultado del reconocimiento.

Caso de uso 3. Gestionar accesos.

Actor: usuario.

1. El usuario pulsa sobre gestión de accesos y se muestran todos los accesos almacenados.
2. El sistema muestra las distintas opciones relacionadas con los accesos.

Caso de uso 3.1. Buscar por matrícula.

Actor: usuario.

1. El sistema muestra un buscador donde el usuario tendrá que introducir un patrón de búsqueda.
2. Al aceptar se mostrarán todos los accesos que cumplan este patrón.

Caso de uso 3.2. Buscar por fecha.

Actor: usuario.

1. El sistema muestra un buscador por fechas.
2. Al aceptar se muestran todos los accesos entre unas fechas determinadas.

Caso de uso 4. Reentrenar OCR (Optical Character Recognition).

Actor: usuario.

1. El sistema permitirá al usuario reentrenar el modelo encargado del reconocimiento de caracteres dejándole introducir sus propias imágenes.
2. Cuando se acepte el reentrenamiento se preguntará si se desea guardar una copia del antiguo y en cualquier caso se almacenará el nuevo.

Caso de uso 5. Configurar aplicación.

Actor: usuario.

1. Se da al usuario la opción de introducir las dimensiones de la matrícula a reconocer y la posibilidad de acceder a la configuración avanzada.
2. La nueva configuración no será definitiva hasta que no sea aceptada por el usuario, que tendrá la posibilidad de guardar una copia de la configuración antigua.

Caso de uso 5.1. Modificar imágenes de entrada.

Actor: usuario.

1. El usuario tiene la posibilidad de ajustar a su gusto los parámetros de los algoritmos que preprocesan las imágenes que llegan a la aplicación.

Caso de uso 5.2. Mostrar pasos intermedios

Actor: usuario.

1. El usuario tendrá la posibilidad de mostrar los pasos intermedios al resultado final, es decir, ver cada una de las modificaciones por las que va a pasar la imagen antes de ser procesada.

2 REQUISITOS NO FUNCIONALES

2.1 ARQUITECTURA

La aplicación ha de ser funcional en cualquier sistema operativo Windows de 64 bits, especialmente Windows 10, que será el sistema operativo usado durante el desarrollo. No se asegurará el correcto funcionamiento en sistemas operativos antiguos.

El diseño en tres capas hará que todas las funcionalidades de la lógica de negocio sean accesibles desde la interfaz de usuario. Los datos que produzca la aplicación serán almacenados en bases de datos para poder ser consultados en cualquier momento.

2.2 RENDIMIENTO

Al ser el reconocimiento de la matrícula un requisito imprescindible para que se abra la barrera de acceso al parking o a la autopista, el proceso de reconocimiento no debe ser nunca superior a 2 segundos.

2.3 INTERFAZ DE USUARIO

Deberá tener una estructura clara, dejando visibles en los menús todas las funcionalidades de la aplicación. La forma de estos menús se decidirá en la fase de diseño de la interfaz.

Capítulo 4. Diseño

1 Diseño de la interfaz

Teniendo en cuenta que la aplicación va a ser utilizada para realizar demostraciones a clientes, no existe la necesidad de hacer una interfaz tremendamente atractiva visualmente, pero sí debe mostrar toda la funcionalidad que tendremos disponible usando la aplicación.

1.1 Creación del prototipo

Los prototipos de esta aplicación han sido desarrollados mediante la herramienta de prototipado *Balsamiq Mockups 3*. Esta herramienta permite crear prototipos que, aunque visualmente sean simples, son navegables y permiten que nos hagamos una idea de cómo se podrá acceder a la funcionalidad a través de los menús de la interfaz.

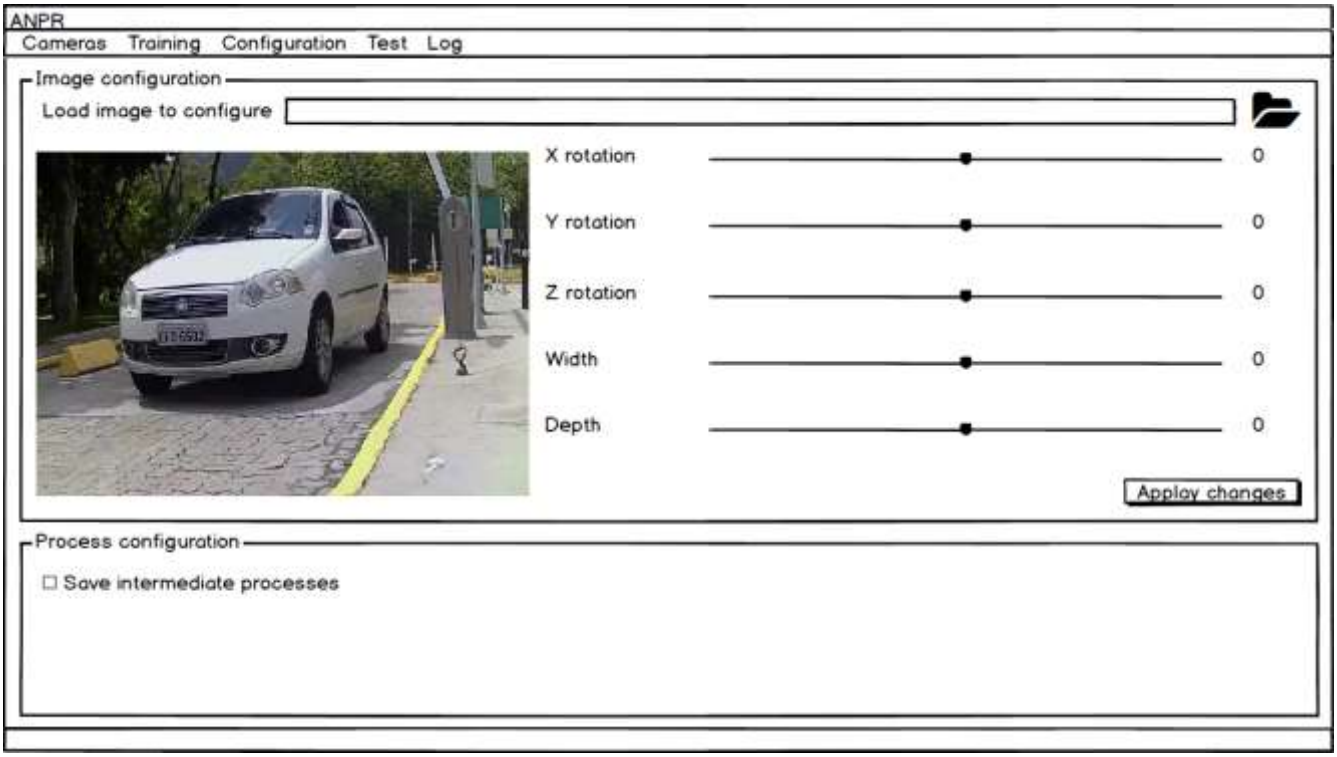

Figura 5. Aspecto de las interfaces del prototipo.

Como he comentado en el párrafo anterior, el prototipo es navegable y, a través de las entradas del menú superior, podremos movernos por la aplicación.

El único inconveniente es que la navegabilidad del prototipo solo es visible a través de la propia aplicación y solo podemos acceder a ella de manera gratuita durante 30 días.

Nota: todas las interfaces del prototipo se podrán encontrar al final de este documento en el anexo correspondiente.

1.2 EVALUACIÓN DEL PROTOTIPO DESARROLLADO

Además de para hacerte una idea de cómo va a estar estructurada la aplicación antes de comenzar con el desarrollo, un prototipo es muy útil para comprobar si esta es o no usable.

Estudiando la facilidad de uso, conseguimos adaptar la aplicación a los usuarios finales para que estos consigan saber usarla rápidamente y sin problemas.

Otro beneficio es que los usuarios finales de la aplicación pueden echar en falta cosas que, seguramente, se puedan añadir con facilidad debido al temprano estado del producto.

Para evaluar la usabilidad de la interfaz, he elegido el método de investigación o indagación. Más concretamente, dentro de este método, utilizaré las entrevistas y la observación de campo como ejes principales de la evaluación del prototipo.

Para que una prueba de usabilidad se realice correctamente, necesitamos preparar los siguientes aspectos:

- **Selección de los evaluadores:** en este caso, como la aplicación se iba a usar dentro de la empresa, me decanté por elegir como evaluadores a los usuarios reales, es decir, mis propios compañeros.
- **Selección de las tareas:** como la funcionalidad de la aplicación no es excesiva, no he pensado ninguna tarea concreta y se probará la funcionalidad de toda la aplicación. Antes de comenzar, se debe dejar claro que no solo hay que centrarse en analizar la funcionalidad pura, también es deseable comentar aspectos de navegabilidad.
- **Selección de datos cualitativos y cuantitativos:** en la planificación se decidió qué resultados se iban a almacenar una vez completado el proceso de reconocimiento. Aquí, los usuarios finales podrán expresar si realmente es suficiente o hace falta recoger algo más.
- **Preparación de los mecanismos de registro:** en este caso voy a optar por las entrevistas personales aprovechando la cercanía y lo fácil que resulta acceder a los evaluadores.

  Las preguntas formuladas en la entrevista deben ser como mínimo las siguientes:
  - ¿Echa en falta alguna funcionalidad?
  - ¿La forma de realizar tests le parece intuitiva?
  - ¿Le parecen correctas las anotaciones de los botones de cada interfaz?
  - ¿Cree que son suficientes los campos almacenados en la base de datos?

Como observador en sus primeras interacciones con el producto, podría añadir, si hay problemas, preguntas como:

- ¿Por qué ha pensado que esta tarea se realizaría de esta forma?
- ¿Qué le ha llevado a hacer esta tarea?

De este modo, podremos enriquecer nuestra aplicación con las aportaciones de los usuarios y no nos quedaremos en el punto de vista del desarrollador.

1.3 RESULTADOS DEL PROCESO DE EVALUACIÓN

El siguiente paso, tras observar a los evaluadores usando el prototipo, es realizar una entrevista con ellos para comentar sus aspectos positivos y negativos.

Entre los aspectos positivos encontramos:

- La navegación entre las distintas pestañas es sencilla e intuitiva.
- Los botones dentro de cada una de las interfaces dejan claro para qué sirven.
- En general, parece útil para hacer demostraciones de cómo funciona el reconocimiento de matrículas.

Por otro lado, se comentaron varios aspectos negativos:

- En el formulario principal (Figura 6), donde supuestamente vemos el video en directo, tenemos la posibilidad de añadir nuevas cámaras, pero una vez tenemos más de una no tenemos la posibilidad de elegir entre ellas.

Figura 6. Video en directo.

- A simple vista no parece haber ninguna zona para mostrar mensajes de error que se pudieran dar al no completar los datos requeridos. Esta falta de retroalimentación se observa sobre todo en las zonas donde el usuario debe realizar ciertas tareas para que el proceso funcione correctamente. Por ejemplo, en la zona de test (Figura 7), el usuario debe añadir el archivo con el entrenamiento de los caracteres y debe cargar una imagen. De no ser así, el procesado fallará. Por este tipo de cosas, se echa en falta una zona de mensajes de error.

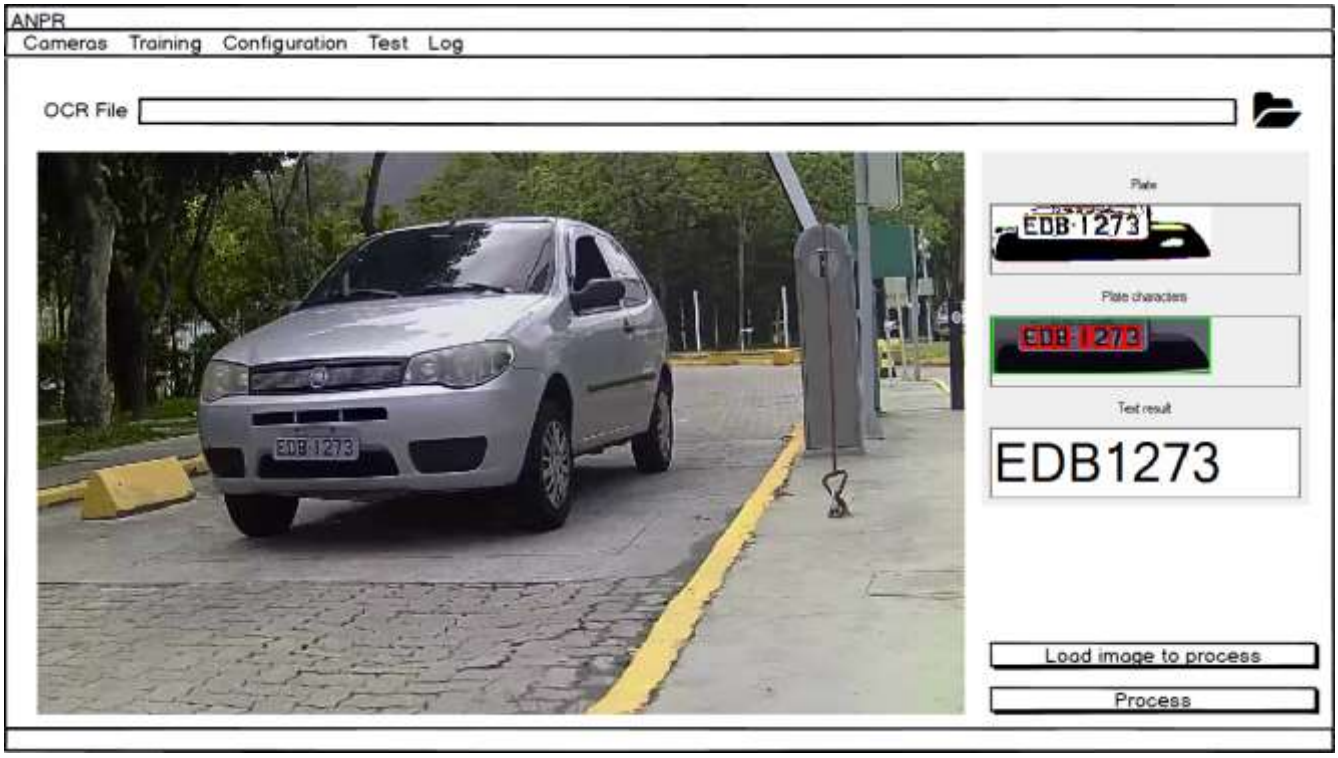

Figura 7. Zona de test.

1.4 CAMBIOS A REALIZAR EN VISTA DE LOS RESULTADOS

Tras valorar la información recibida en la reunión, se decide poner solución a los dos problemas principales planteados de la siguiente forma:

- **Imposibilidad de selección de cámara:** se decide retirar el historial de reconocimientos (parte derecha de la Figura 6) y añadir en su lugar una lista con cada una de las cámaras accesibles por la aplicación.
  Esta lista deberá actualizarse cada vez que se añada o elimine una cámara.
- **Retroalimentación en determinadas acciones:** en la reunión se hizo especial hincapié en la zona de testeo. Por ello, se decide añadir en el espacio entre el

resultado y los dos botones (Figura 7) un campo de texto vacío donde aparecerán los errores cuando los haya.
También se piensa añadir mensajes en la zona de reconocimiento de video y en la zona de entrenamiento del modelo.

- **Añadido de metáforas:** para facilitar la comprensión de determinadas acciones disponibles a través de la interfaz se van a añadir metáforas como las siguientes:

Figura 8. Metáforas de añadir cámara, eliminar cámara y abrir archivo.

2 ORGANIZACIÓN DE LA LÓGICA DE NEGOCIO

Como se ha comentado en el capítulo 2 de este mismo documento, la intención es desarrollar una aplicación de escritorio dividida en capas donde podamos deducir de manera más o menos lógica qué harán cada una de las clases del negocio con solo leer su nombre.

Siguiendo este modelo organizativo se facilitará el mantenimiento del código ya que, si algún tercero tiene que realizar algún cambio sobre el mismo, localizará con más facilidad las zonas a cambiar.

3 SOLUCIÓN DE PERSISTENCIA

Debido a la simplicidad de la base de datos necesaria para guardar los datos deseados, este apartado recibe el nombre de solución de persistencia y no el de diseño de la base de datos.

Para una persistencia tan sencilla se podría incluso usar un simple documento de texto pero, dado que va a haber consultas sobre los datos, decido que es mejor realizar una pequeña base de datos en MySQL. El diagrama relacional resultante de esta base de datos es el siguiente:

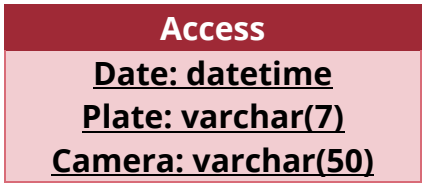

Figura 9. Diagrama de la base de datos

Como se puede observar en el diagrama, solo tendremos una tabla llamada *Access* que almacenará, de cada acceso, los siguientes campos:

- **Date:** de tipo *datetime* que almacenará la fecha y la hora del acceso.
- **Plate:** de tipo *varchar(7)* que almacenará el resultado textual de la matrícula.
- **Camera:** de tipo *varchar(50) almacenará la cámara desde la cual recibimos la imagen.*

Los tres campos formarán la clave primaria de la tabla para evitar así cualquier tipo de duplicado.

Capítulo 5. Implementación

1 PREPARACIÓN DEL ENTORNO DE TRABAJO.

Para poder comenzar a trabajar debemos tener OpenCV funcionando correctamente en el equipo. Para que conseguir el funcionamiento adecuado pueden ser necesarios dos programas adicionales además de la propia librería.

1.1 SOFTWARE NECESARIO

- **OpenCV:** librería libre de visión artificial.
- **MinGW:** implementación de los compiladores GCC de Unix para Windows.
- **CMake:** herramienta multiplataforma de generación automática de código.

1.2 INSTALACIÓN DE MINGW

Descargamos el instalador (mingw-get-setup.exe) desde la siguiente dirección:

https://sourceforge.net/projects/mingw/files/Installer/

Tras completar la instalación aparecerá una lista con los paquetes a instalar. Debemos seleccionar los siguientes:

- mingw32-base
- mingw32-gcc-g++
- msys-base
- mingw-developer-toolkit

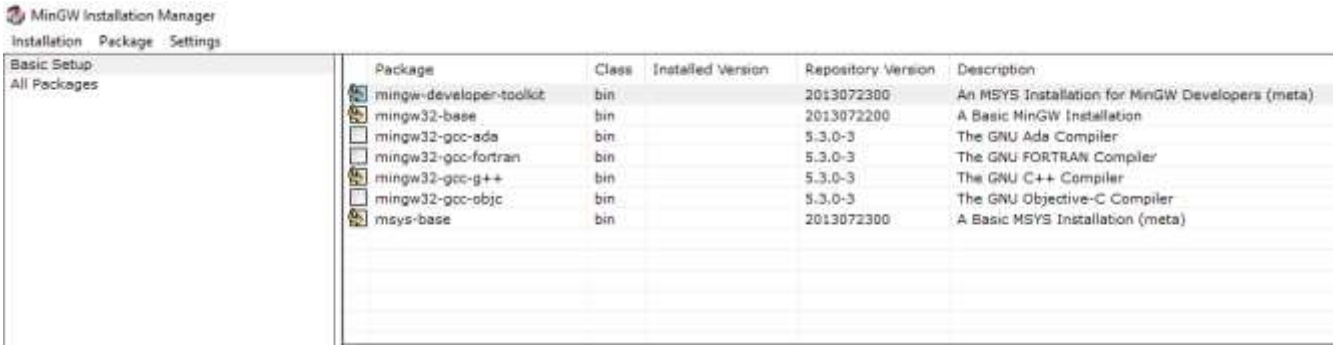

Figura 10. Paquetes a instalar.

Una vez seleccionados, los instalamos pulsando en *Installation* > *Apply changes.*

Para poder compilar OpenCV en Windows, hay que realizar una modificación en el código fuente de un fichero de MinGW. Ese fichero es:

C:\MinGW\include\commctrl.h

Realizar la modificación que se indica en este enlace:

http://answers.opencv.org/question/51987/highgui-window_w32cpp-error-tbbuttoninfo-was-not-declared-in-this-scope/

(Modificar "#if 0" por "#if 1" y "0x0300" por "0x0500" - líneas 13 y 14 del fichero)

Por último, añadimos el directorio C:\MinGW\bin a la variable de entorno PATH. *Sistema* > *Configuración avanzada del sistema* > *Variables de entorno* > *Path* > *Examinar* > *C:\MinGW\bin* > *Aceptar.*

En caso de que se use como IDE Visual Studio en cualquiera de sus versiones (como es el caso de este proyecto) este paso no sería necesario, ya que Visual Studio trae su propio paquete de compiladores para C++.

1.3 INSTALACIÓN DE OPENCV

Para descargar la biblioteca basta con entrar en la página web oficial de los desarrolladores y acceder al apartado *Releases* (http://opencv.org/releases.html).

Tras completar la descarga tendremos que extraer el ejecutable. La extracción es simple: solo hay que pulsar *Extract* en la pantalla que aparece tras arrancar el ejecutable.

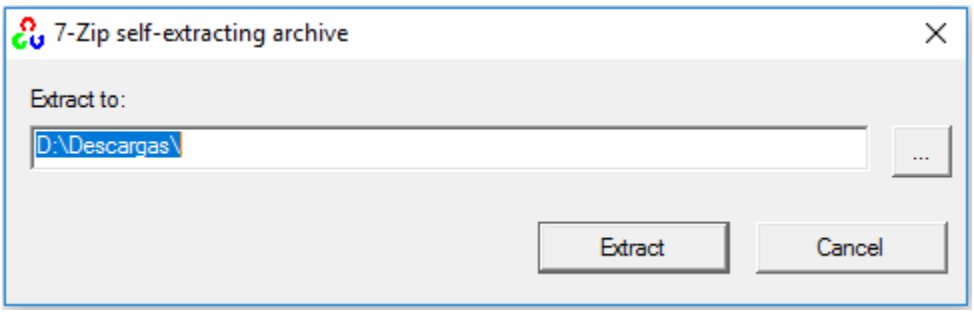

Figura 11. Ventana de extracción de OpenCV.

Una vez extraído, tendremos acceso al siguiente contenido:

- Documentación de la librería.
- Librería precompilada para Java.
- Librería precompilada para Python.
- Librería precompilada para Visual Studio de 32 bits.
- Librería precompilada para Visual Studio de 64 bits.

- Código sin compilar, para poder compilarlo según las necesidades propias. Esto permite también la compilación de ejemplos muy útiles para estudiar el funcionamiento de la librería.

Las librerías precompiladas pueden ser de gran utilidad si queremos comenzar a usar rápidamente la librería, pero si no nos vale con la compilación con los parámetros preestablecidos, tendremos que hacer uso de CMake.

1.4 INSTALACIÓN DE CMAKE

Como se ha comentado anteriormente, este paso solo será necesario si no nos sirve con las librerías precompiladas disponibles tras la extracción de OpenCV.

Descargamos el instalador (cmake-3.8.0-rc1-win64-x64.msi) desde la siguiente dirección:

https://cmake.org/download/

Durante el proceso de instalación debemos asegurarnos de añadir CMake al Path del sistema en la siguiente pantalla:

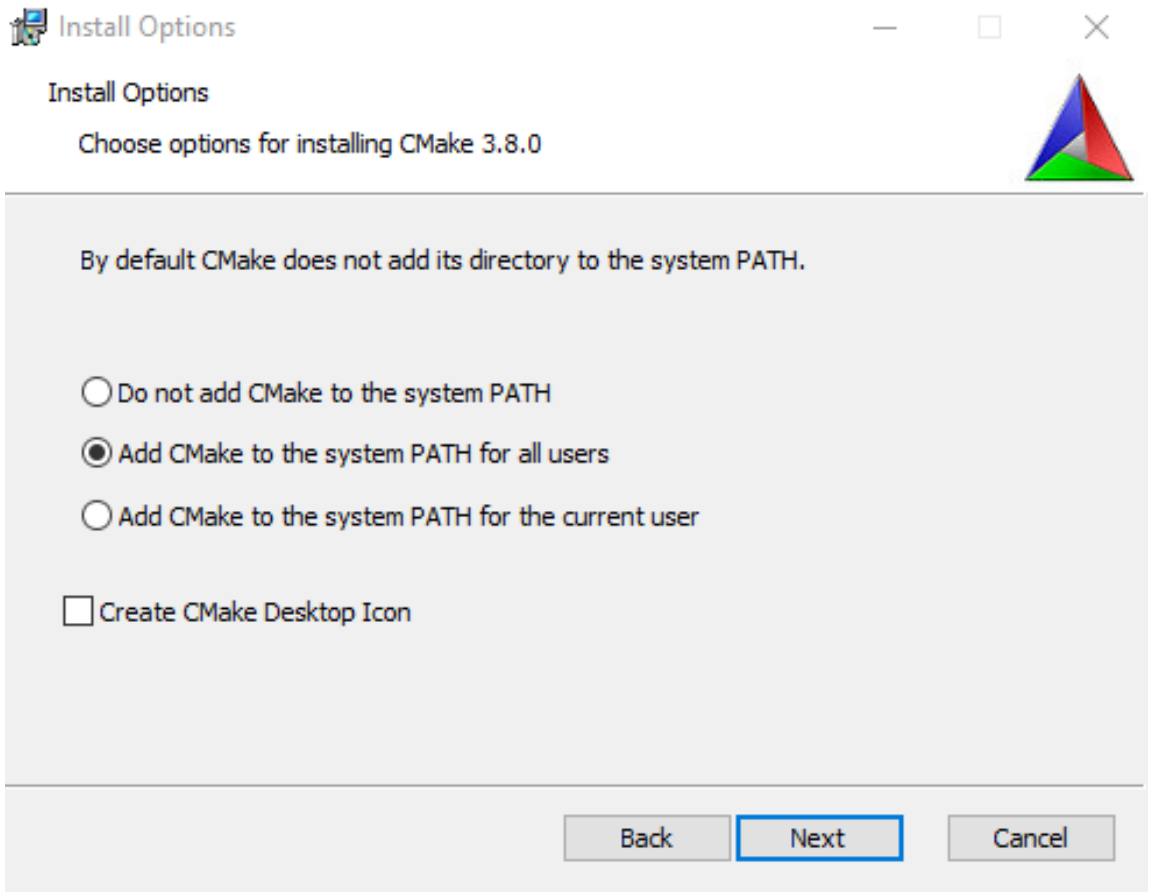

Figura 12. Añadir CMake al path del sistema.

Una vez completada la instalación de CMake lo ejecutamos y elegimos las carpetas origen y destino:

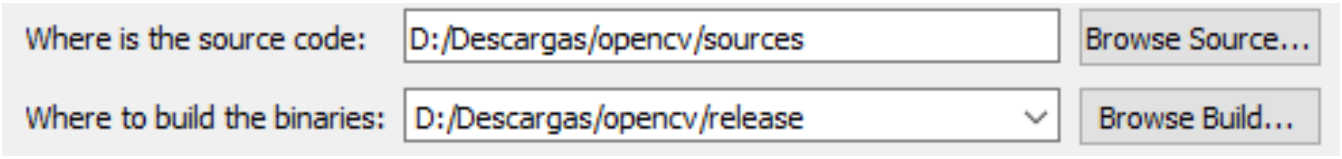

Figura 13. Localización del código fuente y la ruta de compilación.

Una vez elegidas las rutas, pulsamos el botón *Configure*. Esto nos permitirá configurar los archivos de compilación para el entorno de desarrollo que especifiquemos (Netbeans, Codeblocks, Visual Studio...). Tenemos que tener cuidado al elegir la compilación adecuada para 32 o 64 bits.

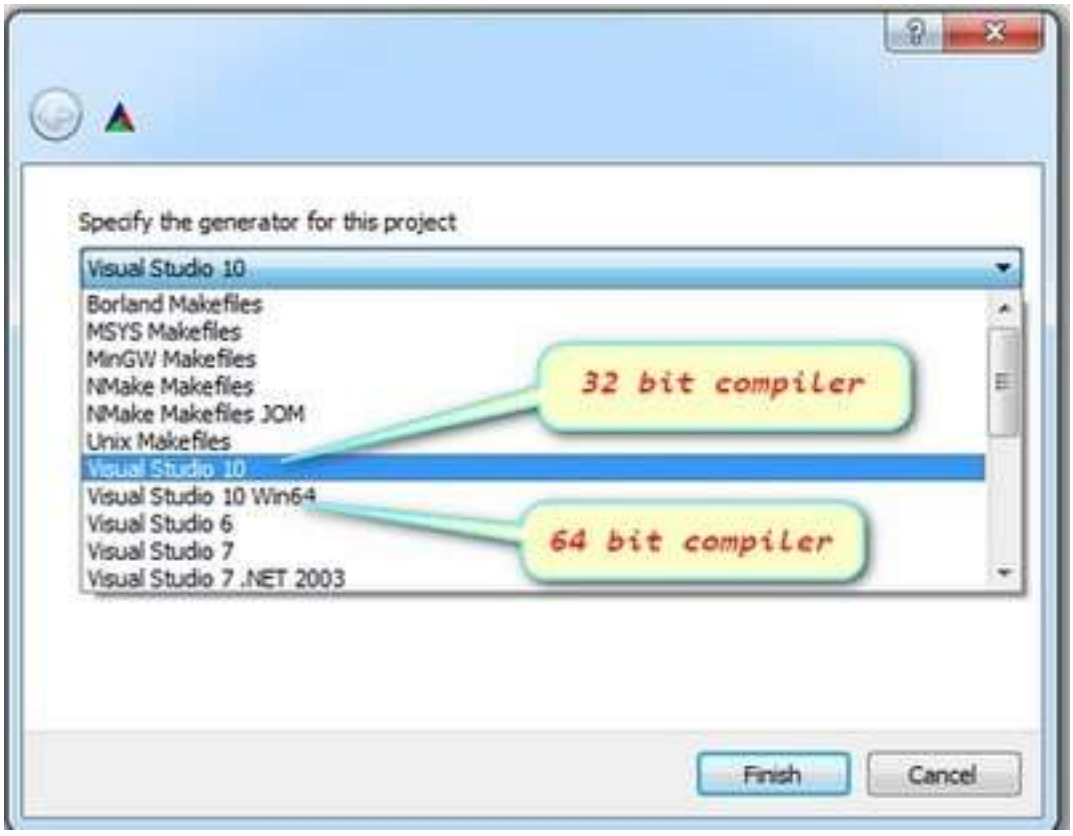

Figura 14. Proyecto de 32 o 64 bits.

Tras elegir el entorno, (en este caso Visual Studio 14 2015 Win64) se completará la configuración y podremos elegir qué partes de la librería generamos para la futura

compilación. Será en este paso en el que podremos elegir si compilar o no los ejemplos o deseleccionar alguna parte de la librería que no vayamos a necesitar.

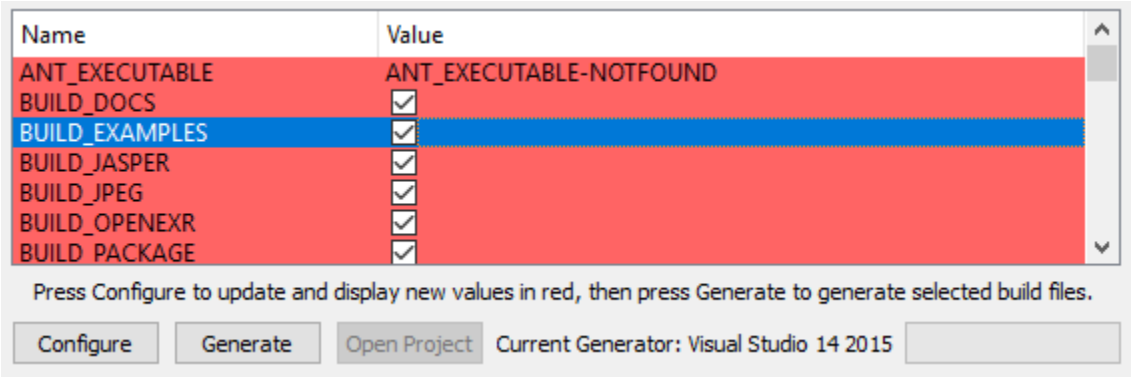

Figura 15. Selección de las partes de la librería a generar.

Para poder visualizar las cámaras en *streaming*, es recomendable marcar la construcción de *Gstreamer.* Para ello, debemos marcar la línea **WITH_GSTREAMER_0_10**.

Cuando se tengan las partes deseadas seleccionadas, volvemos a pulsar *Configure* y después *Generate*.

1.5 COMPILACIÓN

Debido a la gran diferencia que hay entre compilar OpenCV para Visual Studio con la compilación para el resto de entornos de desarrollo, voy a hacer dos apartados en este punto aunque, para este proyecto, solo hace falta seguir el primero de ellos (Compilación para Visual Studio).

1.5.1 *Compilación para Visual Studio*

Abrimos la carpeta en la cual CMake ha generado su código. Si seguimos con la estructura de directorios del ejemplo, la ruta sería **D:\Descargas\opencv\release**. Una vez situados en la ruta correcta, tendremos que abrir la solución llamada **OpenCV.sln**.

Cuando tengamos cargado el proyecto por completo (esperar pacientemente este paso porque si no tendremos errores por cargar de manera incompleta las dependencias), elegimos la opción *Debug* y pulsamos y ejecutamos el proyecto. Cuando acaba esta operación, cambiamos *Debug* por *Release* y repetimos el mismo proceso.

Cuando OpenCV complete la compilación correctamente tanto en *Debug* como en *Release*, buscamos en el explorador de proyectos uno llamado **INSTALL**, hacemos clic derecho

sobre dicho proyecto, buscamos la opción *Solo proyecto* y pulsamos en *Compilar solo INSTALL*. Esta operación la debemos hacer únicamente en modo *Release*.

Por último, añadimos al PATH del sistema el directorio:

%DIR_OPENCV%/release/install/x64/vc14/bin

1.5.2 *Compilación para el resto de entornos de desarrollo*

En concreto, las pruebas para el resto de entornos se han hecho con NetBeans.

En esta ocasión nos tenemos que dirigir a la ruta donde CMake ha generado su código, pero a través de la consola de comandos de Windows (cmd). Por tanto, para realizar esta compilación, debemos ejecutar los siguientes comandos:

- **cd D:\Descargas\opencv\release**
- **mingw32-make.** Como se ha comentado en el punto de instalación de MinGW, serán necesarios estos compiladores, ya que los entornos de desarrollo no incorporan ninguno.

Así comenzará la compilación de OpenCV (proceso que lleva unos 30 minutos). Con todas las modificaciones y procesos anteriormente mencionados realizados correctamente, debería llegar al 100% sin dar errores.

Cuando termine, añadimos a la variable de entorno PATH el directorio:

%DIR_OPENCV%/release/bin.

1.6 CREACIÓN DE UN PROYECTO PARA PROBAR LA INSTALACIÓN EN VISUAL STUDIO

Para incluir las dependencias de OpenCV dentro de Visual Studio, habrá que cambiar las propiedades del proyecto.

Para poder almacenar esta configuración y usarla en distintos proyectos se recomienda usar el administrador de propiedades (Ver>Otras ventanas>Administrador de propiedades). En esta pestaña, podemos añadir propiedades a cada una de las versiones del proyecto, 32 o 64 bits tanto en modo *Release* como en modo *Debug.*

Para añadir una hoja de propiedades, hacemos clic derecho sobre la versión deseada y pulsamos *Agregar nueva hoja de propiedades de proyecto.*

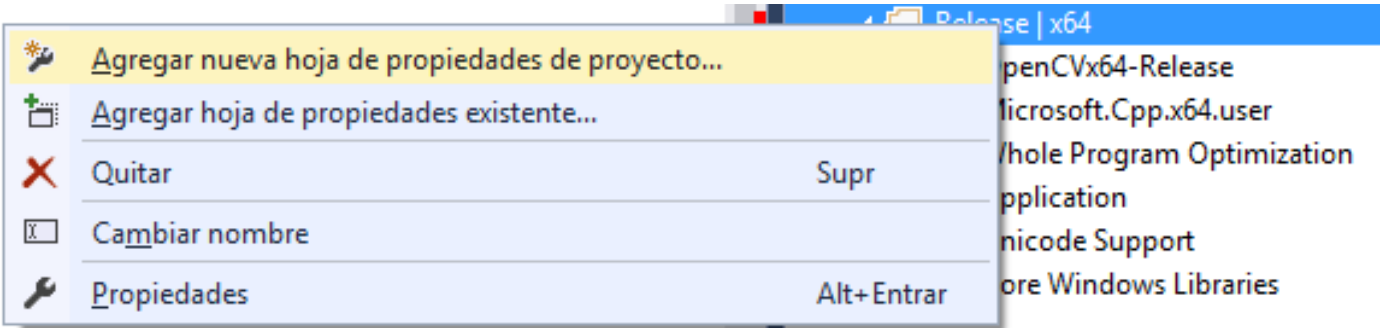

Figura 16. Como añadir nuevas hojas de propiedades.

Una vez creada la hoja entramos en ella y modificamos los siguientes campos:

- **C/C++ > General > Directorios de inclusión adicionales**

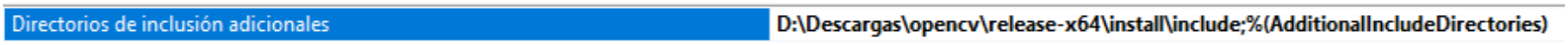

Figura 17. Directorio de inclusión para Release x64.

- **Vinculador > General > Directorios de bibliotecas adicionales**

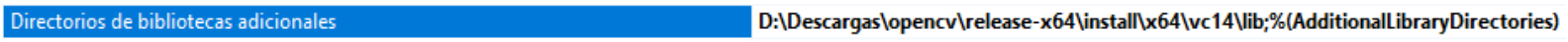

Figura 18. Directorios de bibliotecas adicionales para Release x64.

- **Vinculador > Entrada > Dependencias adicionales**

  Para añadir las dependencias adicionales en Visual Studio, hay que adjuntar en este punto el nombre de cada una de las librerías que se van a utilizar. Este paso nos resultará más sencillo si hemos marcado en CMake la opción **WORLD**, que compilará toda la librería en dos únicos archivos, uno para Debug (acabado en número de versión de OpenCV + d) y otro para Release (acabado en número de versión de OpenCV).

  Si no hemos hecho este paso tendremos unos 20 archivos por modo (release/debug). Copiar cada uno de los nombres puede ser tedioso pero, para facilitar el trabajo, podemos usar la consola de Windows. Si abrimos la consola desde la carpeta lib, generada tras la compilación, y ejecutamos los siguientes comandos, obtendremos los nombre en dos ficheros de texto diferentes:

  - **dir/b *d.lib > libDebug.txt**: para extraer los nombres de Debug
  - **dir/b *13.lib > libRelease.txt**: para extraer los nombrs de Release (donde en el ejemplo aparece un 13 deberemos poner el número de versión de OpenCV).

  Una vez obtenidos los nombres los debemos añadir del siguiente modo:

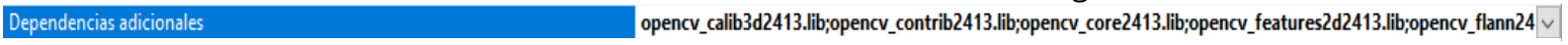

Figura 19. Dependencias adicionales para Release.

Completados todos estos pasos, tendremos solventadas todas las dependencias de OpenCV y podremos probar un código de ejemplo como el siguiente:

```
#include "opencv2/opencv.hpp"
#include "opencv2/highgui/highgui.hpp"

using namespace cv;
using namespace std;

int main(int argc, char** argv) {
	//create a gui window:
	namedWindow("Output", 1);

	//initialize a 120X350 matrix of black pixels:
	Mat output = Mat::zeros(120, 350, CV_8UC3);

	//write text on the matrix:
	putText(output,
		"Hola mundo :)",
		cvPoint(15, 70),
		FONT_HERSHEY_PLAIN,
		3,
		cvScalar(0, 255, 0),
		4);

	//display the image:
	imshow("Output", output);

	//wait for the user to press any key:
	waitKey(0);

	return 0;
}
```

Si todo está configurado según los pasos anteriores, el resultado final debería ser el siguiente:

Figura 20. Resultado final del ejemplo.

2 LIBRERÍAS Y TECNOLOGÍAS UTILIZADAS

En esta sección veremos qué librerías y tecnologías se han utilizado para llevar a cabo el proyecto.

Cualquier cambio con lo expuesto anteriormente en la definición del alcance (Capítulo 2 epígrafe 1.2), quedará explicado en el siguiente apartado.

2.1 OPENCV

OpenCV comenzó su desarrollo en 1999 de la mano de Intel. Desde este momento la librería se orientó hacia la visión artificial con el objetivo de proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esta eficiencia se ha logrado realizando su programación en C y C++ optimizados y aprovechando, si es posible, los procesadores multinúcleo. La eficiencia es total si se usa conjuntamente con un procesador Intel ya que, por herencia de los inicios de su desarrollo, podemos llamar a rutinas específicas de bajo nivel.

Actualmente, tras años de desarrollo, cuenta con más de 500 funciones que abarcan desde los procesos por visión hasta la calibración de cámaras. Además, incluye métodos de aprendizaje automatizado.

Figura 21. Logo de OpenCV

Como es una librería tan extensa no he exprimido al completo su funcionalidad. A continuación, veremos las partes de OpenCV más destacables entre las que he utilizado.

2.1.1 *OpenCV Core*

En esta parte de la librería se concentra la funcionalidad central, es decir, contiene las clases básicas para al tratamiento de imágenes (junto con OpenCV imgproc). El ejemplo más claro es la clase *Mat* que representa un array bidimensional que es utilizado para almacenar imágenes u otro tipo de datos (histogramas, nubes de puntos…)

2.1.2 *OpenCV ML (Machine Learning)*

La biblioteca de aprendizaje automatizado está compuesta por un conjunto de clases y funciones orientadas a la clasificación estadística, la regresión y el agrupamiento de datos.

La mayoría de estos algoritmos se implementan como clases de C++. Como los algoritmos tienen distintas características (capacidad para manejar datos faltantes o entradas de datos categóricos), todas estas clases tienen poco en común. Aun así, toda la funcionalidad común está abstraída y viene definida en *CvStatModel*, la clase padre de todas las clases de OpenCV ML.

2.1.2.1 OCR. Reconocimiento óptico de caracteres

El reconocimiento de caracteres es la parte de mi proyecto en la cual he tenido que utilizar las funciones de aprendizaje automático. Para ello me he apoyado en las máquinas vector soporte (Support Vector Machines). Con estos algoritmos he podido generar un archivo de entrenamiento donde se almacena el aprendizaje y así poder decidir qué carácter tenemos.

2.1.3 *OpenCV Highgui*

Proporcionan los elementos necesarios para poder construir una interfaz de usuario utilizando OpenCV. Gracias a ella es posible cargar imágenes creadas como objetos de OpenCV en una interfaz.

2.2 VISUAL C++ Y VISUAL STUDIO

Visual C++ engloba el desarrollo de aplicaciones en C++ o C en entornos Windows. Viene integrado en el entorno de desarrollo de Microsoft Visual Studio.

Visual C++ también incluye las bibliotecas para el desarrollo de interfaces para Windows, por lo que además de para la lógica de la aplicación, lo he utilizado para el desarrollo de la interfaz, que en un principio iba a desarrollarse en C# (este cambio está justificado en el Capítulo 7 apartado 2)

2.3 MYSQL

Es un sistema de gestión de bases de datos relacionales cuyo desarrollo comenzó en 1995 en el seno de una empresa con el mismo nombre y que fue adquirido en 2010 por Oracle.

Actualmente es considerado como el sistema gestor de bases de datos *open source* más popular del mundo.

2.4 DIRENT.H

Esta librería permite gestionar los directorios de manera sencilla. Además, el tratamiento de los directorios es multiplataforma, es decir, que esta misma librería se podría compilar con los compiladores de Unix, algo muy importante si se pretende hacer la lógica multiplataforma.

3 ESTRUCTURA DEL PROYECTO

En un principio tenía pensado estructurar las capas de la aplicación en tres proyectos distintos, uno por capa. Esta forma de trabajo es la que había utilizado a lo largo de la carrera con las tecnologías de Microsoft.

Al intentar hacer llamadas a, por ejemplo, el proyecto de la lógica desde el proyecto de la interfaz había cientos de errores de dependencias. Tras estar algo más de dos días revisando el proyecto, buscando posibles soluciones en foros y haciendo gran cantidad de pruebas, no fui capaz de encontrar solución a este problema. Por ello, decidí hacer la separación de capas de forma lógica y no física. Es decir, todas las capas están en el mismo proyecto, pero nombradas de forma que se puedan diferenciar con facilidad.

3.1 INTERFAZ

El único punto de acceso a la interfaz de usuario es la pantalla correspondiente al reconocimiento con cámaras. A partir de ahí podremos movernos por la aplicación y acceder al resto de funcionalidades.

Para dejar claro el funcionamiento de la interfaz podemos ver como navegar por ella en el siguiente diagrama de navegación.

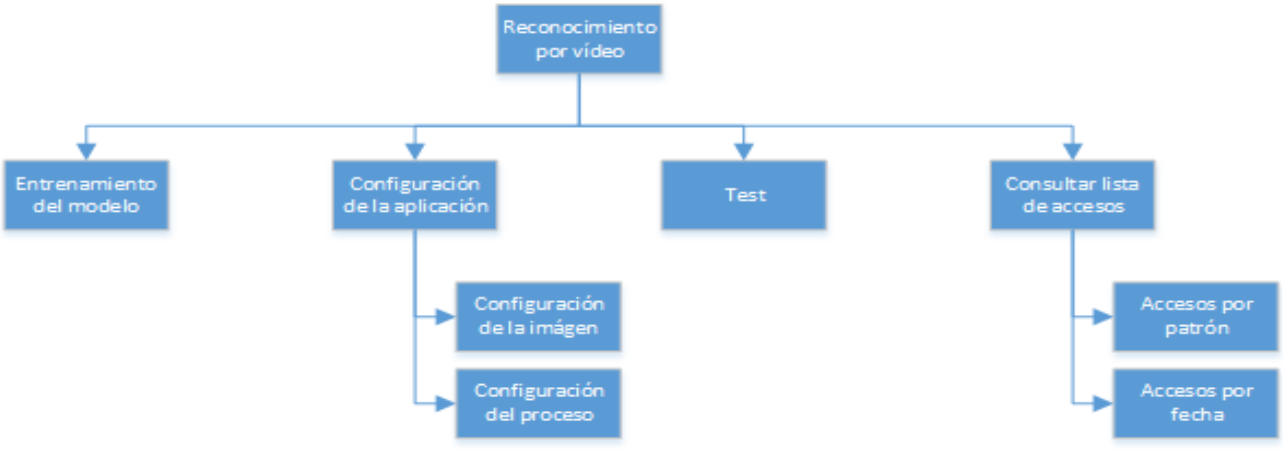

Figura 22. Diagrama de navegación

3.2 LÓGICA

La lógica de la aplicación se estructura en cuatro grandes bloques:

- **Lógica de procesado de XML:** apoyándome en la clase XML de Visual C++, desarrollo una serie de métodos que permiten añadir o eliminar nodos de un fichero XML.
- **Lógica de procesado de directorios:** envuelve la funcionalidad de la anteriormente comentada librería Dirent.h. De esta forma puedo procesar con facilidad directorios de Windows.
- **Lógica con funciones auxiliares:** a su vez dividida en cuatro clases:
    - Lógica con los métodos necesarios para cargar objetos de OpenCV en un interfaz de Windows Forms.
    - Lógica para convertir Strings (cadenas de texto) en vectores de caracteres.
    - Lógica para convertir números enteros en cadenas de texto.
    - Lógica para procesar los objetos de imágenes de OpenCV (Mat), pudiendo rotarlos o desplazarlos con solo hacer llamadas a estos métodos.
- **Lógica de reconocimiento:** engloba todos los métodos necesarios para crear modelos de reconocimiento y para procesar las imágenes y devolver la matrícula en formato textual.

3.3 PERSISTENCIA

Debido a la sencillez de la base de datos una única clase se encarga de gestionar las búsquedas en la base de datos y el proceso de añadido de nueva información. Para la búsqueda hay dos métodos: uno recibe una fecha y otro un patrón en formato textual y para añadir información solo hay uno que recibe los dos parámetros a almacenar, ya que la fecha se genera en el momento del almacenado.

Capítulo 6. Pruebas del sistema

A continuación, se presentan todas las pruebas realizadas una vez desarrollada la aplicación, los resultados obtenidos y los cambios realizados cuando ha sido necesario.

1 PRUEBAS UNITARIAS

Se han llevado a cabo una serie de pruebas unitarias en las cuales solo se hacía uso de la funcionalidad a probar.

1.1 RECONOCIMIENTO

1.1.1 *Reconocimiento a través de cámaras*

Prueba:	**Reconocimiento a través de cámaras**
Descripción:	La persona que esté realizando el test enfoca en la cámara una imagen de un coche y pulsa la tecla Enter.
Resultado:	Se muestra la captura sobre la que se está realizando el reconocimiento y la interfaz se rellena con el resultado del mismo.
Comentarios:	En ocasiones hay caracteres como la 'I' y el 1 que sufren permutaciones debido a las similitudes.

1.1.2 *Reconocimiento a través de test*

Prueba:	**Reconocimiento a través de test**
Descripción:	La persona que esté realizando el test carga un archivo de entrenamiento y una imagen de un coche. Pulsa el botón de procesado situado en la propia interfaz.
Resultado:	La interfaz se rellena con el resultado del reconocimiento.
Comentarios:	En ocasiones hay caracteres como la 'I' y el 1 que sufren permutaciones debido a las similitudes.

1.1.3 *Resolución de las permutaciones de caracteres*

Si una letra o un número se intercambian por otro carácter de su mismo tipo no hay solución posible.

En cambio, si la confusión es entre letra y número o viceversa, nos podemos aprovechar de que la estructura de la matrícula es siempre la misma (NNN LLLL, siendo la N=número y la L=letra). Tras procesar casi 100 imágenes, se llega a la conclusión de que las principales permutaciones son las siguientes:

- El 0 con la 'O'
- El 1 con la 'I'
- El 2 con la 'Z'
- El 5 con la 'S'

- El 7 con la ‘T’
- El 8 con la ‘B’
- El 9 con la ‘P’

Para solucionar este problema, se desarrolla un método de postprocesado del resultado que sí reconoce una de las letras anteriores en las tres primeras posiciones la cambia por el número correspondiente y si reconoce un número en las cuatro últimas posiciones lo cambia por una letra.

```
string applayCountryRules(string licensePlate) {
	stringstream ss
	int i = 0
	if(licensePlate.size() != 7){
		return licensePlate;
	}
	for(const char& c : licensePlate){
		if(i < 3){
			switch(c){
			case ‘0’: ss << ‘O’; break;
			case ‘1’: ss << ‘I’; break;
			case ‘2’: ss << ‘Z’; break;
			case ‘5’: ss << ‘S’; break;
			case ‘7’: ss << ‘T’; break;
			case ‘8’: ss << ‘B’; break;
			case ‘9’: ss << ‘P’; break;
			default: ss << c; break;
			}
		}
		else{
			switch(c){
			case ‘O’: ss << ‘0’; break;
			case ‘I’: ss << ‘1’; break;
			case ‘Z’: ss << ‘2’; break;
			case ‘S’: ss << ‘5’; break;
			case ‘T’: ss << ‘7’; break;
			case ‘B’: ss << ‘8’; break;
			case ‘P’: ss << ‘9’; break;
			default: ss << c; break;
			}
		}
		i++

	}
	return ss.str();
}
```

Este método procesa el resultado devuelto por el método de reconocimiento y lo modifica según las reglas anteriormente enunciadas para finalmente devolver el reconocimiento modificado.

1.2 CONFIGURACIÓN DEL PROCESADO

Prueba:	Configuración del procesado
Descripción:	La persona que esté realizando la prueba selecciona los ítems deseados de la configuración del procesado y comprueba su efecto.
Resultado:	Los resultados de la aplicación cambian en función de los ítems marcados.
Comentarios:	-

1.3 ENTRENAMIENTO DEL MODELO

Prueba:	Entrenamiento del modelo
Descripción:	La persona que esté realizando la prueba deberá elegir una carpeta de caracteres estructurada de la forma indicada y pulsa el botón para entrenar el modelo.
Resultado:	Se podrá en funcionamiento un cronometro que además de indicarnos el tiempo de entrenamiento cuando este termine, evita la sensación de que la aplicación no está haciendo nada.
Comentarios:	-

1.4 VISUALIZACIÓN DE LOS DATOS

1.4.1 *Búsqueda por patrón*

Prueba:	Búsqueda por patrón
Descripción:	La persona que esté realizando la prueba introduce un patrón en el buscador y pulsa la lupa.
Resultado:	Se mostrarán las entradas de la base de datos que coincidan con dicho patrón de búsqueda.
Comentarios:	-

1.4.2 *Búsqueda por fecha*

Prueba:	Búsqueda por fecha
Descripción:	La persona que esté realizando la prueba selecciona una fecha desde el selector de fechas y pulsa la lupa.
Resultado:	Se mostrarán las entradas de la base de datos que coincidan con la fecha indicada.
Comentarios:	-

1.5 PRUEBA DE RENDIMIENTO

Prueba:	Prueba de rendimiento
Descripción:	Se realizarán varios reconocimientos para ver si se cumple el requisito no funcional relacionado con el rendimiento.
Resultado:	El reconocimiento deberá tardar menos de 2 segundos.
Comentarios:	En ocasiones, el primer reconocimiento tarda un tiempo muy cercano a los dos segundos, pero a partir de este momento, los reconocimientos son más rápidos.

2 PRUEBAS DE INTEGRACIÓN

Se llevará a cabo una única prueba de integración en la cual se configurará la aplicación con los parámetros deseados, se lanzará un reconocimiento y se comprobará su almacenamiento en la base de datos.

Aunque es una única prueba de integración se lleva a cabo en varios equipos con distintos sistemas operativos y configuraciones. Así la prueba cubrirá más casos que si solo se realizan en el equipo de desarrollo.

Con ello comprobaremos lo siguiente:

- Si los cambios realizados en la configuración se almacenan correctamente y surten efecto en el proceso de reconocimiento.
- Si el proceso de reconocimiento funciona correctamente.
- Si el resultado procesado se almacena correctamente en la base de datos.

Prueba:	**Prueba de integración**
Descripción:	En esta prueba se realizarán todas las tareas expuestas en la descripción anterior (cambiar la configuración, lanzar un reconocimiento y comprobar la base de datos).
Resultado:	La base de datos tendrá una nueva entrada con el resultado del proceso de reconocimiento.
Comentarios:	El resultado obtenido es el esperado en todos los equipos menos en los que no tienen webcam

Tras esta prueba me doy cuenta que cuando en el XML de las cámaras aparece como primera opción la webcam, como es el caso del archivo con la información de las cámaras usado para las pruebas, si el equipo no tiene webcam, el programa es incapaz de cargar la interfaz de usuario porque ocupa todos sus recursos en buscar una webcam.

Para solucionar este problema se desarrolla un método auxiliar que antes de intentar cargar la webcam comprueba si existe y, de no ser así, carga la interfaz aunque no se vea ningún tipo de video.

Capítulo 7. Seguimiento y control

A continuación, se presenta toda la información correspondiente al seguimiento y control del proyecto.

1 SEGUIMIENTO

Al comenzar el Trabajo de Fin de Grado, creé un plan de seguimiento en la herramienta proporcionada por la universidad.

Plan de Seguimiento

Fecha de creación/actualización del plan de seguimiento: 13-02-2017

Fecha de inicio del TFG: 01-02-2017

Fecha de finalización del TFG: 17-05-2017

Nº de horas totales planificadas: 300

Punto de control	Fecha	Nº de horas empleadas
1	15-02-2017	50
2	15-03-2017	150
3	15-04-2017	250
4	15-05-2017	300

Figura 23. Plan de seguimiento.

Esta planificación ha sido fácil de seguir debido a que he desarrollado todo el Trabajo de Fin de Grado dentro de la empresa que me lo ofreció trabajando en él 5 horas diarias.

Hasta el desarrollo de la interfaz de usuario, los tiempos invertidos en cada fase del proyecto iban según lo planificado (*Tabla 1. Estimación de tiempos*) a partir de este punto se produjeron desviaciones horarias y de calendario que se detallarán en el siguiente apartado.

2 COMPARATIVAS DE TIEMPO Y JUSTIFICACIONES DE DESVÍOS

A continuación, se muestran la *Tabla 2* y el *Gráfico 1* donde se pueden ver las diferencias entre el tiempo estimado y el tiempo planificado.

Fase del proyecto	Número de horas estimadas	Número de horas reales
Definición del alcance y planificación temporal	5	5
Preparación del entorno de trabajo	15	15
Diseño de la interfaz de usuario	20	15
Desarrollo de la interfaz de usuario	20	60
Diseño de la lógica de negocio	40	30
Desarrollo de la lógica de negocio	80	100
Diseño de la capa de persistencia	15	5
Desarrollo de la capa de persistencia	20	20
Ensamblado y pruebas	30	15
Reuniones	5	3
Seguimiento del proyecto	10	15
Memoria y documentación	30	35
Diseño de la presentación del proyecto	10	-
Total	**300 horas**	**318 horas**

Tabla 2. Comparativa horas estimadas y reales.

Gráfico 1. Comparativa horas estimadas y reales.

Los desvíos, tanto horarios como de calendario, han sido causados por los siguientes motivos:

- **Preparación del entorno de trabajo:** el día 7 de febrero entré de nuevo a la empresa a 5 horas diarias. Esta tarea lleva 4 días de calendario, lo que deberían ser 20 horas, pero son 15 debido a que uno de los días se pasó instalando Visual Studio y todos sus componentes.

- **Desarrollo de la interfaz de usuario:** como estaba previsto empezó a desarrollarse en C#, pero al intentar integrar la lógica de la aplicación en C++ hubo problemas. Por este motivo decidí pasarme a Visual C++. Este cambio se realizó tras diseñar parte de la lógica (del 6 al 10 de marzo)
  Otro problema que surgió relacionado con el desarrollo de la interfaz fue el uso de los MdiContainers (contenedores de formularios de aplicación). En un principio no los usaba, pero me percaté de que su uso mejoraría notablemente la navegabilidad de la aplicación.
- **Diseño de la capa de persistencia:** dada la simplicidad final de la base de datos el número de horas estimado ha sido muy superior al real.
- **Diseño de la presentación del proyecto:** aparece vacío actualmente debido a que en la fecha de entrega de este documento la presentación no se ha construido.
- **Memoria y documentación:** esta parte ha sufrido un desvío de calendario bastante grande. El motivo de este desvío ha sido haber comenzado a trabajar 8 horas diarias.
- **Reuniones:** mientras escribo este documento, el tiempo dedicado a reuniones es menor al previsto, pero se espera dedicar más tiempo una vez entregado y en el momento del diseño de la presentación del proyecto. Por lo que es de suponer que se llegará al número de horas planificadas al inicio del proyecto.
- **Desarrollo de la lógica de negocio:** en esta fase se incluye la investigación de cómo hacer un OCR con OpenCV, cómo visualizar cámaras y su posterior implementación. Las 20 horas de desvío de la planificación se deben principalmente a los problemas surgidos con la visualización de cámaras IP en OpenCV por los que tuve que recompilar la librería repetidas veces hasta dar con la configuración adecuada.

Aun así, y teniendo en cuenta los desvíos la evolución de las fases del TFG, ha ido según lo previsto. A falta de completar un 20% del producto (valor que le he asignado a este documento) las fases han quedado de la siguiente manera:

- **Planificación:** del 01-02-2017 al 13-02-2017
- **Análisis:** del 5-02-2017 al 15-03-2017
- **Diseño:** del 13-02-2017 al 03-05-2017
- **Producto:** del 20-02-2017 hasta el depósito de este documento, fechado entre el 21 y el 23 de junio.

La comparativa de fechas de inicio y finalización reales y estimadas se puede ver en la siguiente tabla.

Fase del proyecto	Inicio estimado	Fin estimado	Inicio real	Fin real
Definición del alcance y planificación	01/02/2017	06/02/2017	01/02/2017	06/02/2017
Preparación del entorno de trabajo	07/02/2017	10/02/2017	07/02/2017	10/02/2017
Diseño de la interfaz de usuario	13/02/2017	17/02/2017	13/02/2017	15/02/2017
Desarrollo de la interfaz de usuario	20/02/2017	24/02/2017	16/02/2017	28/04/2017
Diseño de la lógica de negocio	27/02/2017	10/03/2017	24/02/2017	06/03/2017
Desarrollo de la lógica de negocio	13/03/2017	07/04/2017	13/03/2017	12/04/2017
Diseño de la capa de persistencia	10/04/2017	24/04/2017	24/04/2017	24/04/2017
Desarrollo de la capa de persistencia	25/04/2017	02/05/2017	25/04/2017	28/04/2017
Ensamblado y pruebas	03/05/2017	12/05/2017	28/04/2017	03/05/2017

Figura 24. Comparativa entre fechas de inicio y finalización planificadas y reales.

Que algunas de las fases no aparezcan en esta figura y sí en la *Figura 3* se debe a que dichas fases no se han finalizado en el momento de la escritura de este documento.

Capítulo 8. Conclusiones

Al reflexionar sobre los problemas surgidos durante el desarrollo del TFG, queda claro que el más grave ha sido la mala elección de tecnologías y la mala elección de elementos a la hora de crear la interfaz. Esto podría haberse evitado si, antes de desarrollar gran parte de la interfaz, se hubiera hecho una prueba de integración entre la lógica en C++ y con OpenCV y la interfaz en C#.

1 APRENDIZAJE INDIVIDUAL

El desarrollo del TFG me ha permitido reforzar mis conocimientos de C++ y adquirir nuevos conocimientos acerca de tratamiento de imágenes y aprendizaje automatizado gracias al uso de la librería OpenCV.

Además, me ha permitido poner en práctica lo aprendido sobre la gestión de proyectos.

2 AMPLIACIONES Y CONTINUACIÓN

Pese a haber cumplido con los objetivos iniciales fijados y haber creado una aplicación que permite demostrar a un cliente cómo funciona el reconocimiento, desde un primer momento se pensó en continuaciones del proyecto. Son las siguientes:

- **Paso de la lógica de la aplicación a Unix.** De este modo se pretende crear una que permita llamar al sistema de reconocimiento desde una consola de un sistema Unix. Para facilitar este paso, el desarrollo de la lógica de la aplicación Windows se ha realizado, dentro de lo posible, en C++ estándar.
- **Internacionalización del reconocimiento.** Crear un sistema que permita cargar archivos de configuración con las dimensiones de matrículas de otros países. De este modo tendríamos tantas opciones de reconocimiento como archivos de configuración y podríamos usar la aplicación en distintos países.
- **Pruebas en un entorno real.** Durante el desarrollo del proyecto no se ha tenido acceso a cámaras situadas en un entorno real de funcionamiento. Todas las pruebas se han hecho o bien con una cámara IP situada en la propia oficina o con la webcam de los portátiles e intentando reconocer fotografías de vehículos en lugar de vehículos reales.

Capítulo 9. Bibliografía

A continuación, se presenta la bibliografía utilizada para el desarrollo de este TFG:

- **Documentación general:**
  - https://www.codeproject.com/
  - https://stackoverflow.com/
  - https://www.wikipedia.org/
- **OpenCV**
  - http://docs.opencv.org/2.4/
  - http://answers.opencv.org/questions/
- **Visual C++**
  - https://social.msdn.microsoft.com/Forums/vstudio/en-US/home?forum=vcgeneral
  - http://www.cplusplus.com/forum/windows/
- **MySQL**
  - https://dev.mysql.com/doc/connector-cpp/en/connector-cpp-apps-windows-visual-studio.html
  - https://dev.mysql.com/doc/
- **Historia de la visón artificial**
  - https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial
  - https://en.wikipedia.org/wiki/Computer_vision
  - https://visartblog.wordpress.com/2013/05/02/resumen-historia-de-la-vision-artificial/
- **Apuntes de asignaturas**
  - Apuntes sobre requisitos, análisis, diseño y pruebas de la asignatura Ingeniería del Software.
  - Apuntes sobre la creación de interfaces de usuario en Windows de la asignatura Diseño de Interfaces de Usuario.